Polo Aquático Português

João Carlos Machado Rocha Pires (up201806079@fe.up.pt)
Luís Rafael Fernandes Mendes Afonso (up201406189@icbas.up.pt)
Luísa Fernandes Lameiro (up200406067@fe.up.pt)

Bases de Dados, 2019/20 - Turma 7, Grupo 701
Mestrado Integrado em Engenharia Informática e Computação
Faculdade de Engenharia da Universidade do Porto

Descrição

Pretende-se armazenar dados relativos ao Polo Aquático português.

Para cada época, há um conjunto de clubes inscritos na federação. Esses clubes, identificados pelo nome, ano de fundação, região (Norte, Centro, Sul ou Ilhas) e número de títulos, poderão inscrever uma ou mais equipas. Para cada uma dessas equipas, deverá ser indicado o escalão para a qual está inscrita, bem como a listagem dos atletas que a constituem, a classificação obtida na época anterior e a classificação atual.

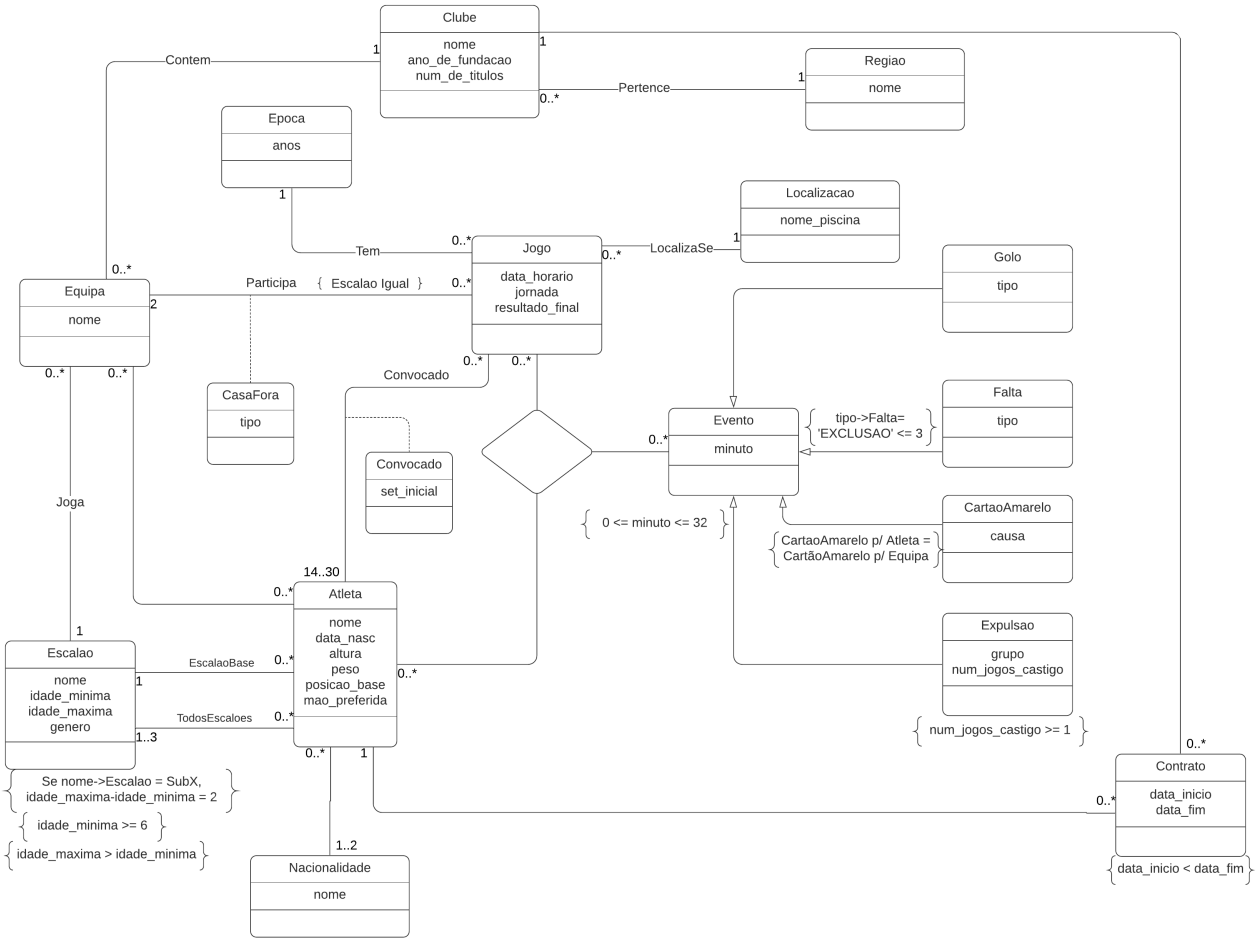
No que ao escalão diz respeito, este deverá conter o nome pelo qual é conhecido, a informação do intervalo de idades permitidas e se se trata de um escalão masculino ou feminino. Quanto à classificação, deverá ser armazenado o total de pontos, a posição na tabela classificativa, os golos marcados e sofridos e o número de jogos efetuados.

Para cada atleta, deverá ser armazenado o nome, a data de nascimento, a nacionalidade, a altura e o peso, a posição em que joga, a mão preferida (esquerda, direita ou ambidestro) e o histórico dos clubes em que já jogou. Este histórico deverá conter, para cada clube, o nome do mesmo e as datas de início e fim.

Para cada época e para cada escalão, existe um conjunto de jogos. Cada jogo tem uma data, indicação da jornada respetiva, local, horário e equipas participantes, bem como o resultado final.

Deverão ainda ser armazenados dados relativos à expulsão de um jogador, se tal ocorrer, identificando o minuto, a causa e o jogador expulso, assim como o número de jogos de castigo atribuídos.

Diagrama UML Revisto



Esquema Relacional

Escalao(idEscalao, nome, idade_minima, idade_maxima, genero)

Atleta(idAtleta, nome, data_nasc, altura, peso, posicao_base, mao_preferida,EscalaoBase → Escalao)

Nacionalidade(idNacionalidade, nome)

Equipa(idEquipa, nome, idClube → Clube, idEscalao → Escalao)

TodosEscaloes(idEscalao → Escalao, idAtleta → Atleta)

AtletaEquipa(idEscalao → Escalao, idAtleta → Atleta)

NacionalidadeAtleta(idNacionalidade → Nacionalidade, idAtleta → Atetla)

CasaFora(idEquipa → Equipa, idJogo → Jogo, tipo)

Jogo(idJogo, data_horario, jornada, resultado_final, idEpoca → Epoca, idLocalizacao → Localizacao)

Convocado(idAtleta → Atleta, idJogo → Jogo, sete_inicial)

Epoca(idEpoca, anos)

Clube(idClube, nome, ano_de_fundacao, num_de_titulos, idRegiao → Regiao)

Regiao(idRegiao, nome)

Localizacao (idLocalizacao, nome_piscina)

Golo (idEvento → Evento, tipo)

Falta (idEvento → Evento, tipo)

CartaoAmarelo(idEvento → Evento, causa)

Evento(idEvento, minuto)

Expulsao(idEvento → Evento, grupo, num_jogos_castigo)

Contrato(idContrato, data_inicio, data_fim, idAtleta → Atleta, idClube → Clube)

AtletaJogoEvento(idAtleta → Atleta, idEvento → Evento, idJogo → Jogo)

Análise Dependências Funcionais e Formas Normais

Não existem violações, quer à *Boyce-Codd Normal Form*, quer à *3NF*, uma vez que, para todas as dependências funcionais identificadas, os atributos do lado esquerdo são ou a chave primária da relação, ou uma chave alternativa para essa relação.

idEscalao → nome, idade_minima, idade_maxima, genero

idAtleta → nome, data_nasc, altura, peso, posicao_base, mao_preferida, idEscalao

idNacionalidade → nome

idEquipa → nome, idClube, idEscalao

idEquipa → idJogo, tipo

idJogo → data_horario, jornada, resultado_final, idEpoca, idLocalizacao

idAtleta, idJogo → sete_inicial

idEpoca → anos

idClube → nome, ano_de_fundacao, num_de_titulos, idRegiao

idRegiao → nome

idLocalizacao → nome_piscina

idEvento → tipo

idEvento → causa

idEvento → minuto

idEvento → grupo, num_jogos_castigo

idContrato → data_inicio, data_fim, idAtleta, idClube

Lista e Forma de Implementação das Restrições

Ao contrário da Entrega 1, nesta segunda parte do trabalho, por recomendação da Professora, decidimos acrescentar mais restrições, quer ao Diagrama UML, quer ao ficheiro ***criar.sql***. Algumas restrições foram apenas adicionadas ao UML, outras apenas ao ***criar.sql*** e outras ainda aos dois, em simultâneo. De seguida listar-se-ão todas as restrições utilizadas, organizadas pelas classes onde as mesmas foram colocadas.

- Classe Escalao

  Nesta classe, incluímos três restrições. Uma delas foi apenas adicionada ao UML. Restringe o intervalo de idades, ou seja, a diferença entre a idade máxima e a idade mínima do escalão, para 2 se se tratar de um escalão da formação, cujo nome será no formato 'SubX', onde X é um número entre 14, 16 e 18.

  $$\texttt{Se nome} \rightarrow \texttt{Escalao = SubX, idade\_maxima-idade\_minina = 2}$$

  As outras duas restrições foram adicionadas tanto no UML como no ***criar.sql***. No entanto, uma delas sofreu uma ligeira alteração.

  $$\texttt{idade\_minima >= 16}$$

  $$\texttt{idade\_maxima > idade\_minima}$$

  Como se pode verificar no seguinte código, a idade mínima passou a ser 16 dado que o ficheiro ***povoar.sql*** apenas insere nesta tabela dois escalões, Sub20 e Seniores, para os quais a idade mínima é 16 (salvo excepções).

```
create table Escalao    (id INTEGER PRIMARY KEY,
                         nome VARCHAR(8),
                         idade_minima NUMBER,
                         idade_maxima NUMBER,
                         genero CHAR(1),
                         UNIQUE(nome, genero),
                         CHECK(idade_maxima>idade_minima
                         and idade_minima>=16));
```

- Associação Participa

  A restrição nesta associação foi apenas colocada no Diagrama UML. Indica que as duas equipas que se defrontam num jogo terão de pertencer ao mesmo escalão.

  ```
  Escalao Igual
  ```

- Classe Falta

  Nesta classe, a restrição foi também adicionada apenas no Diagrama UML. Indica que o número de faltas do tipo Exclusão apenas pode acontecer, no máximo, 3 vezes para cada atleta. Ao fim das 3 faltas do tipo Exclusão, o atleta fica impossibilitado de continuar a jogar naquele jogo, podendo contudo jogar no jogo imediatamente a seguir, diferenciando-se assim de uma Expulsão.

  $$\texttt{tipo} \rightarrow \texttt{Falta = 'EXCLUSAO' <= 3}$$

- Classe CartaoAmarelo

  Tal como nos dois casos anteriores, foi adicionada apenas no UML uma restrição que indica que, se um jogador for sancionado com um Cartão Amarelo, esse mesmo cartão aplica-se a toda a equipa. Tal significa que um próximo cartão do mesmo tipo, ainda que possa ser dado a outro jogador da mesma equipa, implica duplo Amarelo, ou seja Vermelho. Dito de outra forma, usando um exemplo, imaginemos que o jogador nº 7 vê um Cartão Amarelo. De seguida, o jogador nº 3 comete uma falta também para Cartão Amarelo. O jogador com o nº 3 leva Cartão Vermelho dado que é o seu segundo amarelo (o primeiro Cartão Amarelo não lhe foi mostrado, mas foi a um colega da sua equipa).

  ```
  CartaoAmarelo p/ Atleta = CartaoAmarelo p/ Equipa
  ```

- Classe Evento

  Na classe Evento, adicionamos uma restrição, tanto no UML como no ***criar.sql***, que indica que o minuto pode tomar valores entre o intervalo de 0 a 32, ambos inclusive. Isto porque um jogo de Polo Aquático se divide em 4 períodos, cada um com 8 minutos de tempo útil.

```
create table Evento     (id INTEGER PRIMARY KEY,
                         minuto NUMBER,
                         CHECK(minuto>=0 and minuto<=32));
```

$$0<=minuto<=32$$

- Classe Expulsao

  Na classe Expulsao, adicionamos uma restrição para que o número de jogos de castigo atribuídos seja sempre igual ou superior a 1. Qualquer expulsão implica pelo menos o cumprimento de um jogo de suspensão, podendo ser adicionado um número de jogos de suspensão extra em função do grupo no qual a expulsão se encontra.

```
create table Expulsao(evento INTEGER REFERENCES Evento ON UPDATE CASCADE ON
    DELETE SET NULL,
                      grupo CHAR(2),
                      num_jogos_castigo NUMBER DEFAULT 1,
                      PRIMARY KEY(evento),
                      CHECK(num_jogos_castigo>=1));
```

$$num\_jogos\_castigo >= 1$$

- Classe Contrato

  Para a classe Contrato, adicionamos uma restrição que garante que a sua data de fim é posterior à sua data de início.

```
create table Contrato(id INTEGER PRIMARY KEY,
                      data_inicio TEXT,
                      data_fim TEXT,
                      atleta INTEGER REFERENCES Atleta ON UPDATE CASCADE ON
   DELETE SET NULL,
                      clube INTEGER REFERENCES Clube ON UPDATE CASCADE ON
   DELETE SET NULL,
                      CHECK(data_inicio<data_fim));
```

```
data_inicio < data_fim
```

- Classe Epoca

  Na Classe Epoca, para garantir que o ano de início era um valor, no mínimo, realista, adicionamos uma restrição apenas no ***criar.sql*** para garantir que o mesmo era superior a 1900.

```
create table Epoca      (id INTEGER PRIMARY KEY,
                         anoInicio NUMBER UNIQUE NOT NULL
                         CHECK(anoInicio>=1900));
```

```
anoInicio >= 1900
```

- Classe Jogo

  Na Classe Jogo, adicionamos uma restrição que obriga a que a Jornada não só seja positiva como igual ou superior a 1.

```
create table Jogo   (id INTEGER PRIMARY KEY,
                     data_horario TEXT,
                     jornada NUMBER CHECK(jornada>=1),
                     resultado_final VARCHAR(5) DEFAULT '0-0',
                     epoca INTEGER REFERENCES Epoca ON UPDATE CASCADE ON
    DELETE SET NULL,
                     localizacao INTEGER REFERENCES Localizacao ON UPDATE
    CASCADE ON DELETE SET NULL);
```

```
jornada >= 1
```

- Classe Clube

  Na Classe Clube, adicionamos uma restrição dupla, que obriga a que o ano de fundação de um Clube seja igual ou superior a 1900, tal como o ano de início para uma Época (definido na Classe Epoca), e que o número de títulos seja igual ou superior a 0.

```
create table Clube  (id INTEGER PRIMARY KEY,
                     nome TEXT UNIQUE,
                     ano_de_fundacao NUMBER,
                     num_de_titulos NUMBER DEFAULT 0,
                     id_regiao INTEGER REFERENCES Regiao ON UPDATE CASCADE
    ON DELETE SET NULL,
                     CHECK(ano_de_fundacao>=1900 and num_de_titulos>=0));
```

$$ano\_de\_fundacao >= 1900$$

$$num\_de\_titulos >= 0$$

- Classe Atleta

  Finalmente, para a Classe Atleta, adicionamos duas restrições, para o peso e para a altura de cada jogador, não permitindo que sejam inferiores a 30 e a 150, respectivamente.

```
create table Atleta     (id INTEGER PRIMARY KEY,
                         nome TEXT,
                         data_nasc TEXT,
                         altura NUMBER,
                         peso NUMBER,
                         posicao_base TEXT,
                         mao_preferida CHAR(1) DEFAULT 'A',
                         escalao INTEGER REFERENCES Escalao ON UPDATE
   CASCADE ON DELETE SET NULL,
                         CHECK(peso>=30 and altura>=150));
```

$$peso >= 30$$

$$altura >= 150$$

Listagem e explicação das interrogações

- **Interrogação 1** (*int1.sql*)

```
1     SELECT  Atleta.nome as Atleta,
2             Clube.nome as Clube,
3             Contrato.data_inicio as Inicio,
4             Contrato.data_fim as Fim,
5             (case when (cast((cast((julianday(data_fim)-julianday(
    CURRENT_DATE)) as integer)) as integer)<0) then 'Expirado' else 'Activo'
     end) as Validade
6     FROM Contrato, Atleta, Clube
7     WHERE Contrato.atleta=Atleta.id AND Contrato.clube=Clube.id;
8
```

  Explicação em linguagem natural: Retorna uma tabela com o nome do jogador, o clube com o qual tem/teve contrato, a data de inicio e de fim do contrato e se o mesmo já expirou ou se ainda está activo.

  Nota: Para cada jogador, pode haver mais do que um tuplo uma vez que cada jogador pode ter um histórico de mais do que um contrato.

- **Interrogação 2** (*int2.sql*)

```
1     SELECT  Atleta.nome,
2             Atleta.data_nasc,
3             Atleta.altura,
4             Atleta.peso,
5             Atleta.posicao_base as posicao
6     FROM Atleta INNER JOIN AtletaEquipa ON (Atleta.id=AtletaEquipa.atleta)
7     WHERE AtletaEquipa.equipa = 1;
8
```

  Explicação em linguagem natural: Devolve o nome, data de nascimento, altura, peso e posição base de todos os atletas que estão na equipa 1.

- **Interrogação 3** (*int3.sql*)

```
    SELECT DISTINCT Atleta.id, Atleta.nome,
    cast((cast((julianday(CURRENT_DATE)-julianday(data_nasc)) as integer)
   /365.25) as integer) AS Age
    FROM Atleta, Escalao, Nacionalidade
    WHERE   Nacionalidade.nome='Portuguesa'
            AND Age>=20 AND Escalao.genero='M'
            AND Escalao.id=Atleta.escalao;

```

Explicação em linguagem natural: Retorna o ID, nome e idade dos atletas masculinos portugueses com idade igual ou superior a 20 anos.

Utilidade: por exemplo, um seleccionador saber que jogadores pode convocar para um jogo internacional de seniores masculinos.

- **Interrogação 4** (*int4.sql*)

```
    SELECT  Atleta.id as AtletaID,
            Atleta.nome,
            Atleta.posicao_base,
            Equipa.nome as EquipaNome,
            count() as QuantidadeGolos
    FROM    Atleta, Escalao, Convocado, AtletaEquipa, Equipa,
            AtletaJogoEvento, Jogo, Golo
    WHERE   Escalao.id=Atleta.escalao AND
            Escalao.genero='M' AND
            Convocado.atleta=Atleta.id AND
            Convocado.sete_inicial='1' AND
            AtletaEquipa.atleta=Atleta.id AND
            AtletaEquipa.equipa=Equipa.id AND
            AtletaJogoEvento.atleta=Atleta.id AND
            AtletaJogoEvento.jogo=Jogo.id AND
            AtletaJogoEvento.evento=Golo.evento
    GROUP BY Atleta.id
    ORDER BY QuantidadeGolos DESC;

```

Explicação em linguagem natural: Retorna uma tabela com o ID do jogador, o seu nome, a sua posição base, o nome da sua Equipa e a Quantidade de Golos que marcou para todos os jogadores masculinos ordenada pela quantidade descendente de golos

- **Interrogação 5** (*int5.sql*)

```
SELECT Atleta.id, Atleta.nome
FROM Atleta, Convocado, Jogo
WHERE   Atleta.id=Convocado.atleta AND
        Convocado.jogo=Jogo.id AND
        Jogo.id=1 AND
        sete_inicial=1;

```

Explicação em linguagem natural: Devolve o ID e o nome dos jogadores do sete inicial de ambas as equipas de um jogo.

- **Interrogação 6** (*int6.sql*)

```
SELECT  Casa.jornada,
        Casa.data_horario as data,
        Casa.localizacao,
        Casa.club as casa,
        Casa.resultado_final as resultado,
        Fora.club as fora
FROM
   (SELECT *, clube.nome as club
    FROM CasaFora, Jogo, Epoca, ((Equipa, Clube) as Clube)
    WHERE   Jogo.id=CasaFora.jogo AND
            Jogo.epoca=Epoca.id AND
            Epoca.anoInicio='2019' AND
            CasaFora.tipo='C' AND
            CasaFora.equipa=equipa.id AND
            Equipa.clube=clube.id) as Casa,
   (SELECT *, clube.nome as club
    FROM CasaFora, Jogo, Epoca, ((Equipa, Clube) as Clube)
    WHERE   Jogo.id=CasaFora.jogo AND
            Jogo.epoca=Epoca.id AND
            Epoca.anoInicio='2019' AND
            CasaFora.tipo='F' AND
            CasaFora.equipa=Equipa.id AND
            Equipa.clube=Clube.id) as Fora
WHERE Casa.jogo=Fora.jogo
ORDER BY Casa.jornada;

```

Explicação em linguagem natural: Retorna a uma tabela com a jornada, data, localização, a equipa que joga em casa, o resultado e a que joga fora de todos os jogos para a época de 2019.

- **Interrogação 7** (*int7.sql*)

```
SELECT Equipa.id, Equipa.nome as Equipa, COUNT(*) as NumberGoals
FROM Equipa INNER JOIN AtletaEquipa ON (Equipa.id=AtletaEquipa.equipa)
    INNER JOIN Atleta ON (Atleta.id=AtletaEquipa.atleta)
    INNER JOIN AtletaJogoEvento ON (AtletaJogoEvento.atleta=Atleta.id)
    INNER JOIN Golo ON (AtletaJogoEvento.jogo=Golo.evento)
GROUP BY Equipa.id;

```

Explicação em linguagem natural: Devolve o numero total de golos marcados por cada equipa

- **Interrogação 8** (*int8.sql*)

```
SELECT  JogoGolo.id as ID,
        JogoGolo.atleta as Atleta,
        Jogos.Participacoes-JogoGolo.JogosGolos as NoJogosSemGolos
FROM
    (SELECT *, count(*) as JogosGolos
    FROM
        (SELECT Atleta.nome as atleta, Atleta.id as id
        FROM Atleta, AtletaJogoEvento, Golo
        WHERE   Atleta.id=AtletaJogoEvento.atleta AND
                AtletaJogoEvento.evento=Golo.evento
        GROUP BY Atleta.nome, AtletaJogoEvento.jogo
        ORDER BY Atleta.id) as Atletas
    GROUP BY Atletas.id) as JogoGolo,

    (SELECT Atleta.nome as nome,
            Atleta.id as id,
            count(*) as Participacoes
    FROM Atleta, Convocado
    WHERE Atleta.id=Convocado.atleta
    GROUP BY Atleta.id
    ORDER BY Atleta.id) as Jogos
WHERE   Jogos.nome=JogoGolo.atleta AND
        Jogos.id=JogoGolo.id;

```

Explicação em linguagem natural: Devolve o ID, o Nome e o número de jogos em que um determinado jogador não marcou qualquer golo. Nos casos em que esse valor for 0, significa que o jogador marcou em todos os jogos que participou.

- **Interrogação 9** (*int9.sql*)

```
SELECT Clube.nome
FROM Clube, Equipa, Escalao
WHERE   Clube.id=Equipa.clube AND
        Equipa.escalao=Escalao.id AND
        Escalao.nome='SUB20';

```

Explicação em linguagem natural: Devolve o nome dos clubes com equipas de SUB20.

- **Interrogação 10** (*int10.sql*)

```
SELECT CASA.EquipaNome as Equipa, CASA.Pontos+FORA.Pontos as Pontos
FROM
    (SELECT Equipa.nome as EquipaNome,
            Resultado.casa,
            Resultado.fora,
            Resultado.jogo,
            sum(((Resultado.fora<Resultado.casa)*3)+(Resultado.fora=
Resultado.casa)) as Pontos
    FROM
            CasaFora,
            (SELECT substr(resultado_final, 1, pos-1) AS casa,
                    substr(resultado_final, pos+1) AS fora,
                    id as jogo,
                    epoca
            FROM
                (SELECT *, instr(resultado_final,'-') AS pos
                FROM Jogo)) as Resultado,
                Equipa
    WHERE   Resultado.jogo = CasaFora.jogo AND
            Resultado.epoca = 0 AND
            Equipa.id = CasaFora.equipa AND
            CasaFora.tipo='C' AND
            Equipa.nome like '%SUB20'
    GROUP BY Equipa.nome) as CASA,

    (SELECT Equipa.nome as EquipaNome,
            Resultado.casa,
            Resultado.fora,
            Resultado.jogo,
            sum(((Resultado.fora>Resultado.casa)*3)+(Resultado.fora=
Resultado.casa)) as Pontos
    FROM    CasaFora,
            (SELECT substr(resultado_final, 1, pos-1) AS casa,
                    substr(resultado_final, pos+1) AS fora,
                    id as jogo,
```

```
34                          epoca
35                  FROM
36                      (SELECT *, instr(resultado_final,'-') AS pos
37                      FROM Jogo)) as Resultado,
38                      Equipa
39          WHERE   Resultado.jogo = CasaFora.jogo AND
40                  Resultado.epoca = 0 AND
41                  Equipa.id = CasaFora.equipa AND
42                  CasaFora.tipo='F' AND
43                  Equipa.nome like '%SUB20'
44          GROUP BY Equipa.nome) as FORA
45
46      WHERE CASA.EquipaNome=FORA.EquipaNome
47      GROUP BY CASA.EquipaNome
48      ORDER BY Pontos DESC;
49
```

Explicação em linguagem natural: Retorna a tabela classificativa da época 0 do escalão de Sub20.

Nota: A tabela classificativa encontra-se ordenada de forma descendente tendo em conta apenas o número de pontos, pelo que ignora os casos em que a igualdade pontual é igual e se tem de verificar todos os outros critérios, com o confronto directo, o número de golos marcados, etc.

Listagem e explicação dos *triggers*

- **Gatilho 1**

  gatilho1_remove.sql

```
DROP TRIGGER IF EXISTS IdadeEscalao;

```

  gatilho1_adiciona.sql

```
CREATE TRIGGER IdadeEscalao
BEFORE INSERT ON Escalao
WHEN NEW.nome LIKE 'SUB%' AND (NEW.idade_maxima-NEW.idade_minima)<>2
BEGIN
    SELECT RAISE(ABORT, 'Diferenca de idade diferente de 2');
END;

```

  gatilho1_verifica.sql

```
SELECT * FROM ESCALAO;

INSERT INTO Escalao (id, nome, idade_minima, idade_maxima, genero)
            VALUES  (3, 'SUB18', 16, 19, 'M');

```

  Explicação em linguagem natural: Este gatilho, denominado *IdadeEscalao*, não permite que sejam adicionados escalões da formação, ou seja, do tipo SUBX, sendo X um número, tal que a diferença entre a idade mínima e máxima seja diferente de 2. Se tal acontecer, o escalão não é adicionado e é lançada uma mensagem de erro com o seguinte conteúdo: "Diferença de idade diferente de 2".

- **Gatilho 2**

gatilho2_remove.sql

```
DROP TRIGGER IF EXISTS SobreposicaoContrato;

```

gatilho2_adiciona.sql

```
CREATE TRIGGER SobreposicaoContrato
BEFORE INSERT ON Contrato
WHEN EXISTS
     (SELECT *
     FROM Contrato
     WHERE   Contrato.atleta = NEW.atleta AND
             (NEW.data_inicio BETWEEN date(Contrato.data_inicio) AND date(Contrato.data_fim) OR
             NEW.data_fim BETWEEN date(Contrato.data_inicio) AND date(Contrato.data_fim)))

     BEGIN
         SELECT RAISE (ABORT, 'Ja existe um contrato nesse periodo.');
     END;

```

gatilho2_verifica.sql

```
SELECT * FROM Contrato;

-- O inicio do contrato acontece a meio de um contrato ja presente para o mesmo atleta e, ainda que
-- termine apos o fim do contrato ja existente, da erro de sobreposicao
INSERT INTO Contrato (id, data_inicio, data_fim, atleta, clube) VALUES (180, '2015-10-13', '2020-10-11',  0, 0);

-- O inicio do contrato acontece antes do inicio de um contrato ja presente para o mesmo atleta.
-- No entanto, termina durante o periodo de um contrato ja existente para esse mesmo atleta, gerando,
-- novamente, erro de sobreposicao.
INSERT INTO Contrato (id, data_inicio, data_fim, atleta, clube) VALUES (181, '2015-08-13', '2019-09-11',  0, 0);

```

Explicação em linguagem natural: Este gatilho, denominado *SobreposicaoContrato*, tal como o nome indica, não permite a adição de um contrato, para um determinado jogador, se esse novo contrário coincidir com um contrato já existente para o jogador em questão no mesmo período

de tempo. Ou seja, o gatilho verifica se a data de início ou a data de fim do novo contrato a ser introduzido se encontram dentro do período de qualquer contrato associado ao jogador. Se já existir um contrato nesse período, o contrato não é adicionado e é lançada uma mensagem de erro com o seguinte conteúdo: "Já existe um contrato nesse período".

- **Gatilho 3**

gatilho3_remove.sql

```
1    DROP TRIGGER IF EXISTS VerifyPoolAvailability;
2
```

gatilho3_adiciona.sql

```
1    CREATE TRIGGER VerifyPoolAvailability
2    BEFORE INSERT ON Jogo
3    FOR EACH ROW
4    WHEN EXISTS
5        (SELECT *
6        FROM Jogo
7        WHERE   Jogo.localizacao = NEW.localizacao AND
8                Jogo.data_horario BETWEEN datetime(new.data_horario, '-90
   minutes') AND datetime(new.data_horario, '+90 minutes'))
9
10       BEGIN
11           SELECT RAiSE(ABORT, 'Piscina selecionada esta ocupada no
   periodo introduzido');
12       END;
13
```

gatilho3_verifica.sql

```
1    SELECT * FROM Jogo;
2
3    INSERT INTO Jogo (id, data_horario, jornada, resultado_final, epoca,
   Localizacao) VALUES (48, '2019-09-23 20:00', 2, '2-2', 1,1);
4
5    INSERT INTO Jogo (id, data_horario, jornada, resultado_final, epoca,
   Localizacao) VALUES (47, '2019-09-23 18:50', 2, '2-2', 1,1);
6
```

Explicação em linguagem natural: Este gatilho, denominado *VerifyPoolAvailability*, garante que uma piscina só pode ser utilizada para um jogo se não ocorrer outro jogo nos próximos 90 minutos ou se outro jogo começou há menos de 90 minutos. Ou seja, se houve um jogo às 10h, entre as 8h30 e as 11h30 a piscina está indisponível para outros jogos para além do já marcado.